Cindy Li

Nov 14, 2021

IT FDN 100

Assignment 5

https://github.com/cindy-x-li/IntroToProg-Python

# The ToDo List Script

## Introduction

The fifth assignment is to modify a programming template so the user can manage a ToDo List via a menu. The menu guides him or her through a series of options, such as loading previous data, adding, deleting and saving tasks and its priority to a text file. The "ToDo" file will contain two columns of data, "Task" and "Priority." Each pair of data is a Python Dictionary object, and it is loaded into a table of data as a row of data.

## The Script

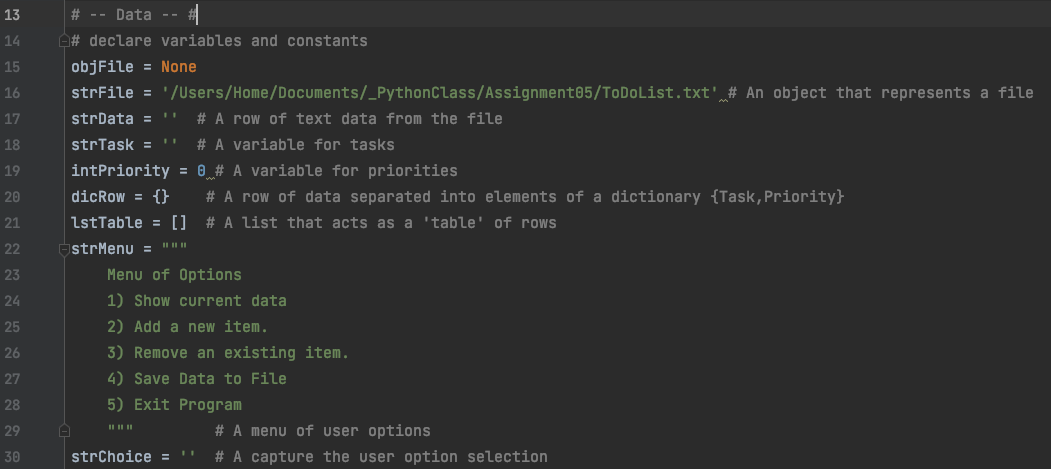
This script differs from previous assignments, because I am using a template where the pseudocode is already written out for me and I am trying out the new concept of separation of concerns as divided up in three following sections: data, procession and presentation.

### Data

To begin, I reviewed the data section and added the following variables (Figure 1):

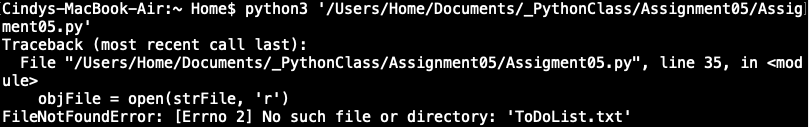
1. Line 16: strFile, for the text file where data will be read from and saved to
2. Lines 18-19: strTask and intPriority, for the user’s input of tasks in a string & priorities in integers

I modified the strMenu by moving the menu options in the presentation section to the data section, since this menu of options are constants (line 22-28). Furthermore, since I created the variable strFile for the text file, I removed it from objFile and set objFile to None (line 15).



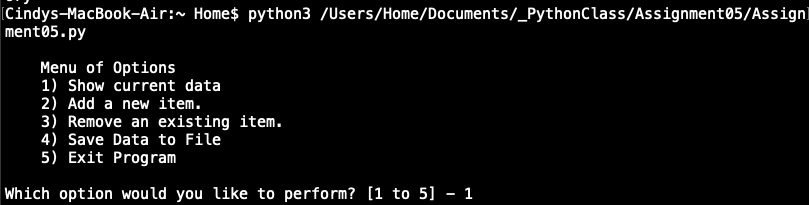
***Figure 1: Data section of the script***

After testing the code in Terminal, the console window for Mac, I put the entire file path for ToDoList.txt, instead of referencing the file by its name. This avoid receiving the FileNotFoundError message (Figure 2).



***Figure 2: FileNotFoundError message in terminal***

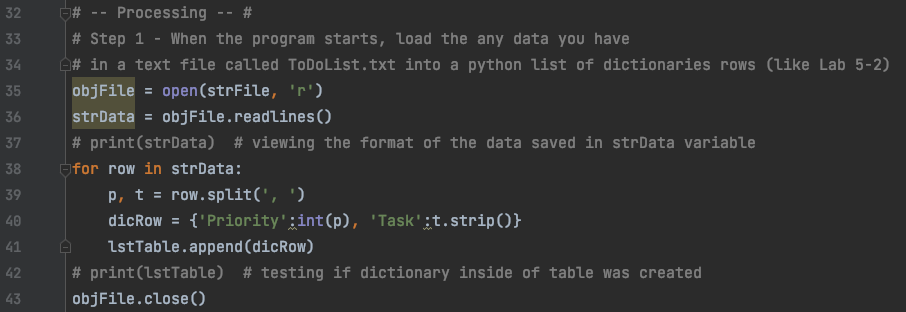
Once I write the entire file path as seen in the data section of Figure 1, the program is able to run smoothly in both Terminal, as seen in Figure 3, and PyCharm, as seen in the following figures. Furthermore, it allows the file to be saved in the current folder as the program. If I move the program, I would have to modify the file path for the text file.



***Figure 3: Script running in terminal***

### Processing

As seen in Figure 4, the file, ToDoList.txt, is opened and any data from the file is loaded into strData (line 35-36). Afterwards, using a for loop to read through each row of strData, each pair of data is loaded into a dictionary via its respective column or key, either as a Priority or Task, by way of unpacking the string into the variables, p and t. They are the values of the dictionary and they are linked with their respective key. Afterwards, the dictionary row is appended to a list (lines 38 – 41). As seen on line 40, I modified the order of the columns to show as priority and task, because I wanted to save the priority as an integer and the strip function is only applicable to strings. If the string is 2nd in the dictionary, then the strip function for the new line in the text file can be used.

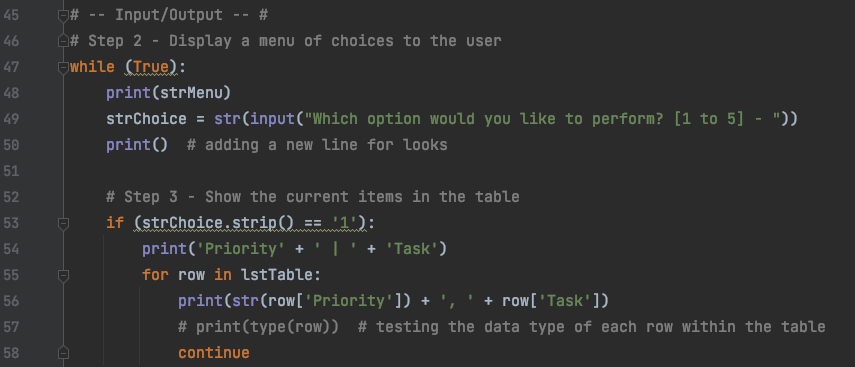


***Figure 4: Processing section of the script***

### Input/Output

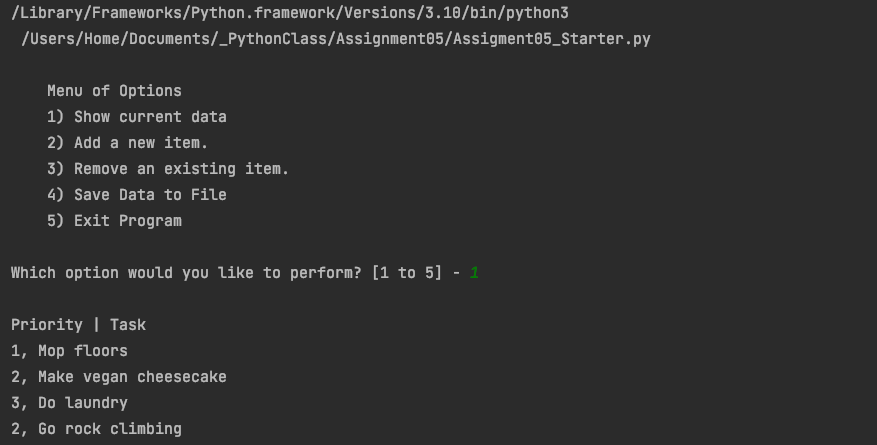
In this section, there is a mixture of processing and I/O, because it is not possible to perform processing statements outside of the if/elif statement without custom functions. This can be seen in the following walkthrough of my code.

For step 2, I used the strMenu variable to print the menu for the user (Figure 5, line 48) and I also request the user to input a number from the menu to proceed with modifying the ToDo List script (Figure 5, line 49).



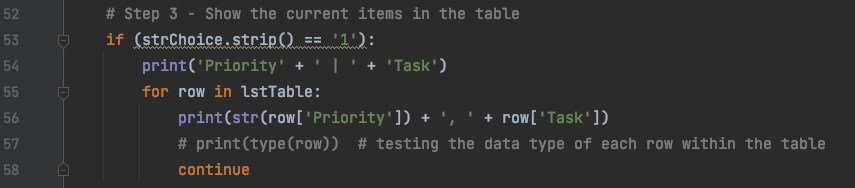
***Figure 5: Output w/ menu options & requesting input from user***

Figure 6 demonstrates the user interface of the script from Figure 3.



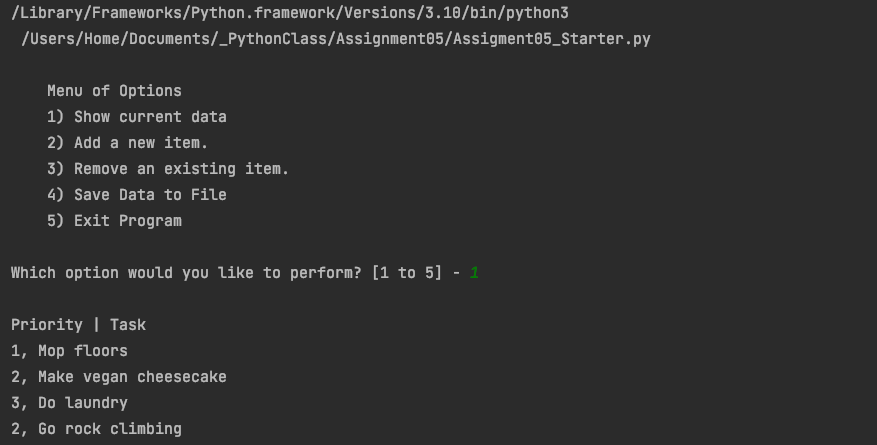
***Figure 6: User interface w/ menu of options and selection request***

If the user chooses option 1, the data within the table is printed out to the user using a for loop, where the values of each dictionary row the values is accessed via the keys, priority and task (Figure 7, lines 55 to 56). Note the first key is an integer, so it is converted to a str in order to use the ‘+’ operator to concatenate the print statement.



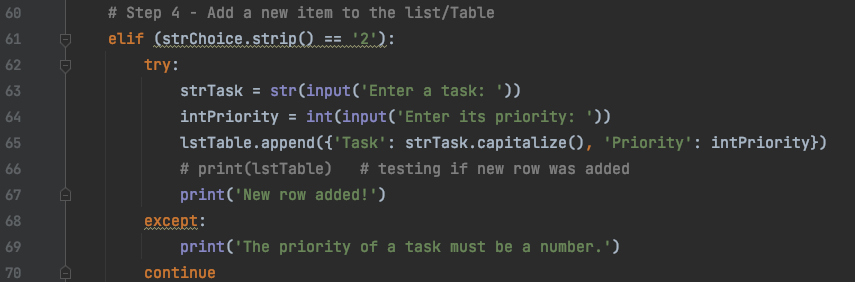
***Figure 7: Script for showing current items in the table***

Figure 8 demonstrates the items from the table are printed out to the user in two columns.



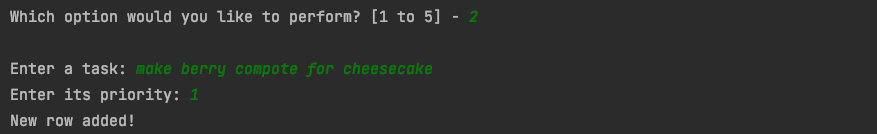
***Figure 8: Items from the ToDo List***

In Step 4, I used the try-except function to ensure the user will input priorities into the ToDo List as integers (Figure 9, lines 62 to 69).

******

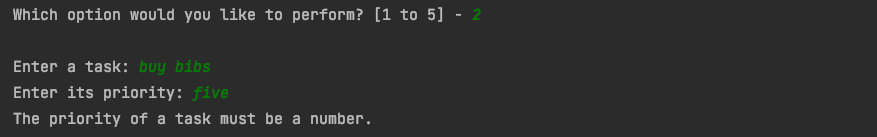
***Figure 9: Script for adding new items to the list/table***

When the user enters a number for the priority, the entry is successfully added to the list/table of data (Figure 10).



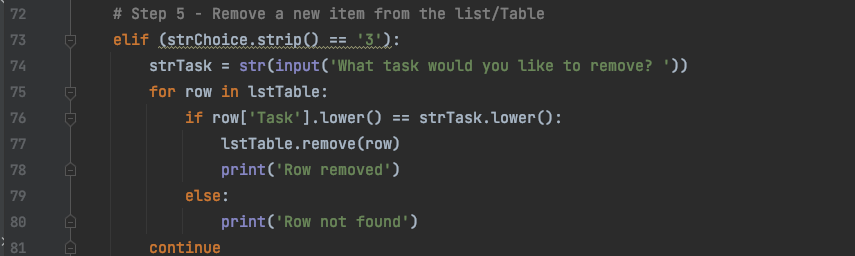
***Figure 10: Demonstrates user adding a new entry***

However, when the user writes out the number for a task’s priority, the entry is rejected and the user is informed enter numbers for the priority in order to put the entry in the ToDo List (Figure 11).

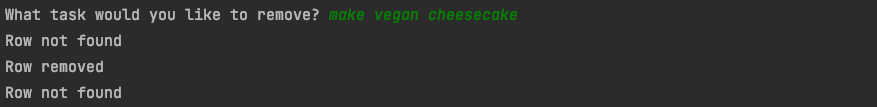


***Figure 11: Demonstrates the user interface result of the try-except function***

Step 5 was the trickiest of all these options to program. As seen in Figure 12, lines 75 to 80, it involved comparing the entered task with all the tasks in the task and removing it if it was the same. If it were not found, the user would be notified. The way the code runs now, for every row it searches in the table, it prints out “Row removed” or “Row not found.” This is does not create a clean user interface (Figure 13). Ideally, these processing statements would return a True or a False for whether or not the task was found. Then that condition determines if the “Row not found” is printed or not.

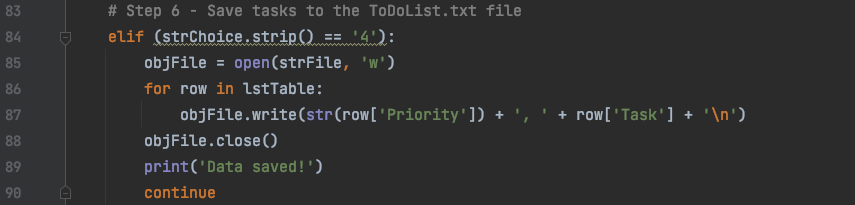


***Figure 12: The script for step 5, removing an item***



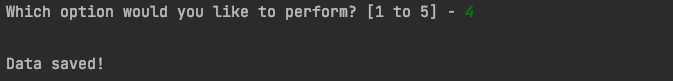
***Figure 13: Removing a task***

The script for step 6 saves the entries to the same file the program originally loaded its data. Here the file is reopened, because a different action needs to be performed on the file. Instead of reading data, the file is opened to write data from the table (Figure 14, line 85). Again, a for loop is used and it writes each line to the file (lines 86-87).



***Figure 14: The script for saving tasks.***

The user is notified when this action is complete (Figure 15)



***Figure 15: The user is notified data is saved.***

Lastly, the user ends the program by entering the number five and Figure 16 demonstrates the code uses a print out a statement to notify the user what they have selected (line 94) and break to leave the while loop in which these options are nested in (line 95).

### Macintosh HD:private:var:folders:yw:13gbt77n1hs35zwm545dn46h0000gn:T:TemporaryItems:Screen Shot 2021-11-13 at 10.48.58 PM.png

***Figure 16: Script for exiting program***

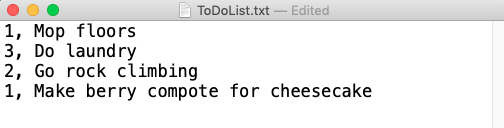
The user see ‘Exiting Program’ when they choose option 5 (Figure 17).

### Macintosh HD:private:var:folders:yw:13gbt77n1hs35zwm545dn46h0000gn:T:TemporaryItems:Screen Shot 2021-11-13 at 7.42.23 PM.png

***Figure 17: Demonstrating how to exit the program***

### The Text File

In Figure 10, a pair of data was entered into the table, and in Figure 13, an entry was removed. These changes can be seen in the resulting saved ToDoList.txt (Figure 18).



***Figure 18: Displaying saved data in a text file***

## Summary

Although it took some time to understand how dictionaries are different from lists, it is now easy to see how they are extremely useful for managing pairs of data. I am looking forward to using functions in the next assignment and making my code more professional and reuseable.